

The Lower The Simpler: Simplifying Hierarchical Recurrent Models

Chao Wang and Hui Jiang

Department of Electrical Engineering and Computer Science
Lassonde School of Engineering, York University
4700 Keele Street, Toronto, Ontario, Canada
{*chwang, hj*}@eecs.yorku.ca

Abstract

To improve the training efficiency of hierarchical recurrent models without compromising their performance, we propose a strategy named as “the lower the simpler”, which is to simplify the baseline models by making the lower layers simpler than the upper layers. We carry out this strategy to simplify two typical hierarchical recurrent models, namely Hierarchical Recurrent Encoder-Decoder (HRED) and R-NET, whose basic building block is GRU. Specifically, we propose Scalar Gated Unit (SGU), which is a simplified variant of GRU, and use it to replace the GRUs at the middle layers of HRED and R-NET. Besides, we also use Fixed-size Ordinally-Forgetting Encoding (FOFE), which is an efficient encoding method without any trainable parameter, to replace the GRUs at the bottom layers of HRED and R-NET. The experimental results show that the simplified HRED and the simplified R-NET contain significantly less trainable parameters, consume significantly less training time, and achieve slightly better performance than their baseline models.

1 Introduction

With the advance of various deep learning frameworks, neural network based models proposed for natural language understanding tasks are becoming increasingly complicated. To the best of our knowledge, a considerable part of these complicated models are both hierarchical and recurrent. For example, Hierarchical Recurrent Encoder-Decoder (HRED) (Sordani et al., 2015; Serban et al., 2016), which is a conversational model, is constructed by stacking three layers of GRUs (Cho et al., 2014). Besides, several well-known Machine Reading Comprehension (MRC) models, such as R-NET (Wang et al., 2017) and Fusion-Net (Huang et al., 2017), are mainly composed of multiple layers of bidirectional GRUs (BiGRUs)

or bidirectional LSTMs (BiLSTMs) (Hochreiter and Schmidhuber, 1997). The above hierarchical recurrent models have achieved excellent performance, but training them usually consumes a lot of time and memory, that is because their computational graphs contain a large amount of operators and trainable parameters, which makes their training computationally expensive.

According to Williams and Zipser (1995), in the training of recurrent neural networks, it is the backward propagation rather than the forward propagation that consumes the majority of the computational resources. Besides, considering the chain rule in the backward propagation, the complexity of computing gradients for a hierarchical recurrent model increases exponentially from the top layer of the model down to the bottom layer. Therefore, to improve the training efficiency of hierarchical recurrent models, our strategy is to **simplify the baseline models by making the lower layers simpler than the upper layers**, which we name as “**the lower the simpler**”. Here “simpler” means containing less operators and trainable parameters. This strategy is guaranteed to work, since it can accelerate the computation of gradients, which is the substance of the backward propagation. However, there is still a big concern: once the baseline models are simplified, will their performance be compromised?

To address this concern, we carry out our proposed strategy to simplify two typical hierarchical recurrent models, namely HRED and R-NET, whose basic building block is GRU. Specifically, we propose Scalar Gated Unit (SGU), which is a simplified variant of GRU, and use it to replace the GRUs at the middle layers of HRED and R-NET. Besides, we also use Fixed-size Ordinally-Forgetting Encoding (FOFE) (Zhang et al., 2015), which is an efficient encoding method without any trainable parameter, to replace the GRUs at the bottom

layers of HRED and R-NET. In the experiments, we separately compare the simplified HRED and the simplified R-NET with their baseline models in terms of both the training efficiency and the performance. The experimental results show that the simplified models contain significantly less trainable parameters, consume significantly less training time, and achieve slightly better performance than their baseline models.

2 Baseline Models

2.1 Hierarchical Recurrent Encoder-Decoder

Hierarchical Recurrent Encoder-Decoder (HRED) is a conversational model for building end-to-end dialogue systems. Since a dialogue is a sequence of sentences, where each sentence is a sequence of words, HRED models this hierarchy with a hierarchical recurrent structure. Specifically, HRED consists of three layers of GRUs, which from bottom to top separately serve as the sentence-level encoder, the dialogue-level encoder, and the decoder. The sentence-level encoder GRU iteratively takes the embeddings of the words in a sentence to update its hidden state, thus its final hidden state is a representation of the sentence. The dialogue-level encoder GRU iteratively takes the representations of the sentences in a dialogue to update its hidden state, thus its hidden state at each time-step is a representation of the current dialogue. The decoder GRU takes the current dialogue representation to initialize its hidden state so as to generate a response sentence word by word.

2.2 R-NET

R-NET is an end-to-end MRC model that predicts an answer span for each given passage-question pair. Specifically, R-NET consists of five layers, which from bottom to top are separately the embedding layer, the encoding layer, the matching layer, the self-matching layer, and the output layer. The embedding layer maps the words to the word-level embeddings and the character-level embeddings. The character-level embeddings are generated by processing the character embeddings of the words with a BiGRU and concatenating the forward GRU final hidden states and the backward GRU final hidden states. The encoding layer processes the concatenation of the word-level embeddings and the character-level embeddings with another BiGRU and concatenates the forward GRU outputs and the backward GRU outputs so as to

generate the context representations. The matching layer uses a gated attention-based BiGRU to fuse the context representations of the question into those of the passage so as to generate the question-aware passage representations. The self-matching layer uses another gated attention-based BiGRU to fuse the question-aware passage representations into themselves so as to generate the final passage representations. On this basis, the output layer uses a pointer network (Vinyals et al., 2015) to generate an answer span.

3 Model Simplification

3.1 Scalar Gated Unit

Just like LSTM, GRU is a recurrent structure that leverages gating mechanisms to capture long-term dependencies in sequential data:

$$\text{Update Gate: } z_t = \sigma(W_z[h_{t-1}, x_t])$$

$$\text{Reset Gate: } r_t = \sigma(W_r[h_{t-1}, x_t])$$

$$\text{New Memory: } \hat{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t])$$

$$\text{Hidden State: } h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

Researchers have proposed several simplified variants of GRU. For example, Zhou et al. (2016) proposed Minimal Gated Unit (MGU), which combines the update gate and the reset gate into a single forget gate. Compared with GRU, MGU contains less trainable parameters, consumes less training time, and achieves similar performance. However, in this paper, to better carry out our proposed “the lower the simpler” strategy, we propose Scalar Gated Unit (SGU), which is an even more simplified variant of GRU:

$$\text{Scalar Update Gate: } z_t = \sigma(w_z[h_{t-1}, x_t])$$

$$\text{Scalar Reset Gate: } r_t = \sigma(w_r[h_{t-1}, x_t])$$

$$\text{New Memory: } \hat{h}_t = \tanh(W_h[r_t * h_{t-1}, x_t])$$

$$\text{Hidden State: } h_t = (1 - z_t) * h_{t-1} + z_t * \hat{h}_t$$

By comparing the formulation of SGU with that of GRU, it is easy to see that both the update gate z_t and the reset gate r_t change from the vectors in GRU to the scalars in SGU. Accordingly, the weights for generating the gates change from the matrices W_z and W_r in GRU to the vectors w_z and w_r in SGU. Besides, the gating operator also changes from the element-wise multiplication \odot in GRU to the scalar multiplication $*$ in SGU. Therefore SGU is guaranteed to be the simplest among all the variants of GRU.

3.2 Fixed-size Ordinally-Forgetting Encoding

Fixed-size Ordinally-Forgetting Encoding (FOFE) is an encoding method that uses the following recurrent structure to map a varied-length word sequence to a fixed-size representation:

$$h_t = \begin{cases} 0, & \text{if } t = 0 \\ \alpha * h_{t-1} + x_t, & \text{otherwise} \end{cases}$$

where h_t is the hidden state at time step t , x_t is the embedding of the t -th word, and α ($0 < \alpha < 1$) is the forgetting factor that decays the previous hidden state. Given a word sequence of length N , the final hidden state h_N is a fixed-size representation of the word sequence. Although formulated as a recurrent structure, FOFE can actually be implemented with an efficient matrix multiplication. Besides, the forgetting factor α is designed as a hyper-parameter so that FOFE contains no trainable parameter. Therefore FOFE is guaranteed to be the simplest among all the recurrent structures. As for the performance, according to Zhang et al. (2015), FOFE based language models outperform their LSTM based competitors.

3.3 Simplified Models

According to the above descriptions, SGU is simpler than GRU, and FOFE is simpler than SGU. Therefore, now we can carry out our proposed “the lower the simpler” strategy by using SGUs and FOFEs to replace certain GRUs in HRED and R-NET. For HRED, we keep the decoder GRU at the top layer unchanged, use a SGU to replace the dialogue-level encoder GRU at the middle layer, and use a FOFE to replace the sentence-level encoder GRU at the bottom layer. For R-NET, we keep the output layer, the self-matching layer, and the matching layer unchanged, use a bidirectional SGU (BiSGU) to replace the BiGRU that generates context representations at the encoding layer, and use a bidirectional FOFE (BiFOFE, i.e., running FOFE both forward and backward) to replace the BiGRU that generates character-level embeddings at the embedding layer. After conducting the above replacements, we finally obtain a simplified HRED and a simplified R-NET.

4 Experiments

4.1 Experimental Datasets

Dialogue Datasets. We compare the simplified HRED with the baseline HRED on two dialogue

datasets, namely MovieTriples (Serban et al., 2016) and Ubuntu (Lowe et al., 2017). MovieTriples contains over 240,000 dialogues collected from various movie scripts, with each dialogue consisting of three sentences. Ubuntu contains over 490,000 dialogues collected from the Ubuntu chat-logs, with each dialogue consisting of seven sentences on average. Both MovieTriples and Ubuntu have been randomly partitioned into three parts: a training set (80%), a development set (10%), and a test set (10%).

MRC Dataset. We compare the simplified R-NET with the baseline R-NET on an MRC dataset, namely SQuAD (Rajpurkar et al., 2016). SQuAD contains over 100,000 passage-question pairs with human-generated answer spans, where the passages are collected from Wikipedia, and the answer to each question is guaranteed to be a fragment in the corresponding passage. Besides, SQuAD has also been randomly partitioned into three parts: a training set (80%), a development set (10%), and a test set (10%). Both the training set and the development set are publicly available, but the test set is confidential.

4.2 Implementation Details

HRED. We implement both the simplified HRED and the baseline HRED with TensorFlow (Abadi et al., 2016). For the word embeddings, we set their size to 200, 400, and 600 on MovieTriples and 600 on Ubuntu, initialize them randomly, and update them during the training. For the forgetting factor α of FOFE, we set it to 0.9 on both MovieTriples and Ubuntu. For the hidden state size of the sentence-level encoder GRU, we set it to 200, 400, and 600 on MovieTriples and 600 on Ubuntu. For the hidden state size of the dialogue-level encoder GRU and SGU, we set it to 1200 on both MovieTriples and Ubuntu. For the hidden state size of the decoder GRU, we set it to 200, 400, and 600 on MovieTriples and 600 on Ubuntu. For model optimization, we apply the Adam (Kingma and Ba, 2014) optimizer with a learning rate of 0.0001 and a mini-batch size of 32. For performance evaluation, we use both perplexity and error rate as evaluation metrics.

R-NET. We implement both the simplified R-NET and the baseline R-NET with TensorFlow. For the word-level embeddings, we initialize them with the 300-dimensional pre-trained GloVe (Pennington et al., 2014) vectors, and fix them during the

Model	Word Embedding	Hidden States (bottom-up)	Trainable Parameters	Training Time (secs * epochs)	Performance (ppl, err rate)
Baseline HRED	200	200-1200-200	10,777,003	4,100 * 33	35.72, 66.62%
	400	400-1200-400	18,740,403	4,660 * 29	34.35, 66.13%
	600	600-1200-600	28,223,803	5,700 * 29	34.11, 65.95%
Simplified HRED	200	200-1200-200	6,456,605	2,030 * 35	35.14, 66.46%
	400	400-1200-400	12,019,605	2,210 * 30	34.01, 66.05%
	600	600-1200-600	18,142,605	2,590 * 29	33.79, 65.89%

Table 1: Comparing the simplified HRED with the baseline HRED on MovieTriples.

Model	Word Embedding	Hidden States (bottom-up)	Trainable Parameters	Training Time (secs * epochs)	Performance (ppl, err rate)
Baseline HRED	600	600-1200-600	40,231,401	51,770 * 33	46.29, 68.76%
Simplified HRED	600	600-1200-600	30,150,203	21,690 * 33	45.55, 68.55%

Table 2: Comparing the simplified HRED with the baseline HRED on Ubuntu.

Model	Trainable Parameters	Training Time (secs * epochs)	Dev Performance (EM / F1)
Baseline R-NET	2,307,991	2454 * 41	71.1 / 79.5
Simplified R-NET	2,007,435	2085 * 38	71.2 / 79.7

Table 3: Comparing the simplified HRED with the baseline HRED on SQuAD.

training. For the character embeddings, we initialize them with the same pre-trained GloVe vectors, and update them during the training. For the forgetting factor α of FOFE, we set it to 0.7. For the hidden state size of both the BiGRUs and the BiSGU, we set it to 128. For model optimization, we apply the Adam optimizer with a learning rate of 0.0005 and a mini-batch size of 32. For performance evaluation, we use both Exact Match (EM) and F1 score as evaluation metrics, which are calculated on the development set.

4.3 Experimental Results

For model comparison in the training efficiency, we use the same hardware (i.e., Intel Core i7-6700 CPU and NVIDIA GeForce GTX 1070 GPU) to train both the baseline models and the simplified models. The experimental results show that our proposed “the lower the simpler” strategy improves the training efficiency of both HRED and R-NET without compromising their performance. On the one hand, as shown in Table 1 and Table 2, the simplified HRED contains 25%–35% less trainable parameters, consumes over 50% less training time, and achieves slightly better perfor-

mance than the baseline HRED. Besides, Table 1 also shows that appropriately scaling up the model brings better performance but consumes more resource, which implies that the simplified HRED will perform better than the baseline HRED when time or memory is limited. On the other hand, as shown in Table 3, the simplified R-NET contains 13% less trainable parameters, consumes 21% less training time, and achieves slightly better performance than the baseline R-NET.

5 Conclusion

In this paper, we propose a strategy named as “the lower the simpler”, which is aimed at improving the training efficiency of hierarchical recurrent models without compromising their performance. This strategy has been verified on two typical hierarchical recurrent models, namely HRED and R-NET, where we replace their middle layers and bottom layers with two simpler recurrent structures. The significance of this paper lies in that it reveals a methodology for avoiding unnecessary complexity in training hierarchical recurrent models, which we believe is applicable to many other hierarchical recurrent models.

Acknowledgments

This work is partially supported by a research donation from iFLYTEK Co., Ltd., Hefei, China, and a discovery grant from Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

- Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.
- Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Hsin-Yuan Huang, Chenguang Zhu, Yelong Shen, and Weizhu Chen. 2017. Fusionnet: Fusing via fully-aware attention with application to machine comprehension. *arXiv preprint arXiv:1711.07341*.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse*, 8(1):31–65.
- Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.
- Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, volume 16, pages 3776–3784.
- Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 553–562. ACM.
- Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.
- Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. 2017. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198.
- Ronald J Williams and David Zipser. 1995. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433.
- Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, and Lirong Dai. 2015. The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 495–500.
- Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. 2016. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234.